Reconstrucción del front-end y back-end de un juez automático de teoría de la computación

Jaime Pascual Torres

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Barcelona 2016

Reconstrucción del front-end y back-end de un juez automático de teoría de la computación

Jaime Pascual Torres

Trabajo Final de Grado
Facultad de Informática de Barcelona
Universidad Politécnica de Cataluña

Presentado por
Jaime Pascual Torres

Barcelona, a 20 de Junio de 2016

Director: Carles Creus
Ponente: Albert Rubio
Director GEP: Xavier Llinars

Índice general

Índice de figuras

Índice de Tablas

En este proyecto hemos reconstruido una aplicación usada por los profesores y alumnos de dos asignaturas, Teoría de la Computación y Compiladores, que se imparten en el Grado de Ingeniería Informática, en la Facultad de Ingeniería Informática de la Universidad Politécnica de Cataluña. La aplicación se llama Racso, se trata de un juez que permite a los alumnos entregar ejercicios y ser evaluados sin necesidad de la atención constante de un profesor.

Este juez requería una serie de cambios por los cuales tomamos la decisión de reconstruirlo. Hemos usado una serie de tecnologías para ayudarnos en esta tarea. Usamos Laravel, un framework PHP basado en Symfony que nos aporta una estructura básica desde la que comenzar, junto con Node y algunas librerías JavaScript para hacer algunas páginas más usables y dinámicas.

Con esta reconstrucción se han mejorado aspectos del Racso tales como la usabilidad de algunas pantallas, se ha separado la lógica que computa las peticiones y la lógica que genera las vistas, haciendo así que sea más fácil añadirle nuevas funcionalidades en el futuro. También se ha creado un nuevo panel de administración para los profesores, desde el cual tienen más facilidades para trabajar con los datos de la aplicación tales como crear nuevos ejercicios, preparar exámenes, o controlar el uso que los alumnos dan de la aplicación. Empezamos por definir una plataforma que sea capaz de soportar la infraestructura, que nos ayude en las tareas de mantenimiento futuras, continuamos por imitar la anterior funcionalidad que se quería conservar y a continuación le añadimos las nuevas funcionalidades.

In this project we wanted to rebuild an application used by teachers and students from two subjects: Theory of Computation and Compilers. These two subjects take place in the Bachelor Degree in Informatics Engineering at the Barcelona School of Informatics from the Polytechnic University of Catalonia. The application is called Racso and it is a judge that allows students to submit exercices and be evaluated without the supervision from their teacher.

This judge required some changes which have motivated us to rebuild it. We used some technologies to help us to do this task. We used Laravel, a PHP framework based on Symfony that provide us a basic structure, Node and some JavaScript packages to make some pages more useful and dynamic.

With this reconstruction some aspects of Racso will be improved, as the usability of some screens and splitted the logic that computes the petition from the logic that generates the views, this way it will be more easy to add new funcionalities in the future. Also we created a new administration panel for the teachers, so they will have more facilities to work with the application data. We started by defining a platform that will be capable of supporting the needed infrastructure and to help us with the future maintenance tasks, then we continued by adding the old features and later the new ones.

En aquest projecte hem reconstruït una aplicació usada pels professors i alumnes de dues assignatures, Teoria de la Computació i Compiladors, que s'imparteixen en el Grau Superior d'enginyeria informàtica en la Universitat Politècnica de Catalunya. L'aplicació s'anomena Racso, es tracta d'un jutge automàtic que permet als alumnes entregar exercicis i ser avaluats sense la necessitat de l'atenció constant d'un professor.

El jutge requeria diversos canvis pels quals vàrem prendre la decisió de reconstruir-lo. Vàrem fer servir un conjunt de tecnologies per ajudar-nos en aquesta tasca. Hem fet servir Laravel, un framework PHP basat en Symfony que ens va ajudar amb una estructura bàsica, junt amb Node i algunes llibreries JavaScript per dotar a les pàgines de més usabilitat i dinamisme.

Amb aquesta reconstrucció es varen millorar aspectes del Racso tals com la usabilitat d'algunes pantalles i es separaren la lògica de còmput de les peticions i la lògica que genera les vistes, fent així que sigui més fàcil afegir-li noves funcionalitats en el futur. També es va crear un nou panell d'administració per als professors, des del qual tindran més facilitats per treballar amb les dades de l'aplicació. Començàrem per definit una plataforma que fos capaç de suportar la infraestructura esmentada i que ens ajudi en les futures tasques de manteniment, continuàrem per imitar l'antiga funcionalitat que es vol conservar per posteriorment afegir-li les noves funcionalitats esmentades abans.

Capítulo 1

Introducción

1.1. Contexto

El sistema educativo se sirve de muchas herramientas para formar a los estudiantes. Muchas de estas herramientas son de carácter electrónico y gran parte se apoyan fuertemente en las tecnologías web. En este campo encontramos todo tipo de editores, terminales controladas, tutoriales guiados. Entre toda esta gama encontramos a los conocidos como jueces. Los jueces suelen ser webs que ofrecen una lista de problemas y permiten a los usuarios/estudiantes enviar propuestas de solución, que serán evaluadas de forma automática.

Gracias a estos jueces, profesores y alumnos pueden mejorar respectivamente con un alto rendimiento ya que los alumnos pueden realizar ejercicios por su cuenta que les ayuden a mejorar y ver en qué fallan sin la constante necesidad de atención por parte de un profesor; y a su vez éste recibe feedback de los avances de los alumnos que le ayudan no solo a establecer el nivel de la clase sino también a detectar los puntos del temario que puedan no haber quedado claros o que necesiten más refuerzo. En conclusión: todos ganan.

En la FIB encontramos diversos jueces, entre ellos, los más famosos el Jutge [1] y el Racso [2]. En este proyecto hemos trabajado sobre el apartado web de

[1] https://www.jutge.org/
[2] https://racso.lsi.upc.edu/juez/

este último: el juez Racso utilizado por los alumnos de Teoría de la Computación y Compiladores.

1.2. El Racso

Desde el punto de vista de un alumno el Racso es una herramienta que corrige ejercicios, con la que podrá entrenar para poder presentarse al examen que hará en el mismo Racso. Algunos alumnos toman el Racso como un reto personal, más personal aún si cabe que el hecho de aprender y aprobar una asignatura. Deciden hacer los ejercicios para poder competir con sus compañeros o por simple afición.

Desde el punto de vista de un profesor el Racso es una herramienta educativa que ayuda a impartir lecciones sobre la teoría de la computación y sobre compiladores, también se basa en la idea de competir y en que el aprendizaje es mayor cuando podemos ver nuestros errores en todo momento. A la vez hay un componente social en el que se pretende abrir o dar la posibilidad a todo el mundo, no solo a los alumnos de la FIB, de estudiar y practicar sobre conceptos de computación, como son las máquinas de estados, autómatas, expresiones regulares, gramáticas, parsing, reducciones entre problemas indecidibles, NP-completos, o reducciones a SAT.

1.3. La organización del documento

En este documento encontraremos la información detallada de como se ha procedido en cada una de las fases del proyecto, desde las fases iniciales donde se estudia previamente las necesidades de la aplicación y toma de requisitos hasta su implementación final, pasando por la gestión del proyecto, las pruebas realizadas y la gestión monetaria.

Empezaremos por definir los objetivos del proyecto y a formular los requisitos como un problema a resolver, en la sección 2. Posteriormente en la sección 3 identificaremos las funcionalidades de la aplicación, tanto a nivel de usuario como a nivel de administrador, parando por las mejoras que vamos a aplicar en la interfaz. Después en la sección 4 hablaremos de la arquitectura del proyecto, los actores implicados que van a utilizar la herramienta y para los cuales se diseñará la aplicación

y que afectará a su estructura. En este apartado también tendremos en cuenta el diseño de la interfaz y la usabilidad de esta misma. Posteriormente en el apartado 5 detallaremos las necesidades de la plataforma donde se va a desarrollar toda la implementación y la plataforma donde va a acabar el producto final. En esta sección también hablaremos de como vamos a efectuar las pruebas, nos centraremos en las pruebas automáticas, que serán muy útiles para futuras mejoras e implementaciones de la aplicación. En la penúltima de las secciones, la 6, hablaremos de toda la gestión del proyecto, la metodología seguida a la hora de desarrollar, la planificación temporal y la gestión económica, así como la sostenibilidad del proyecto y el compromiso social y la identificación de las leyes y regulaciones. Y por último, en la sección 7, defenderemos las competencias técnicas de este trabajo, hablaremos de las conclusiones extraídas durante el proceso y daremos un vistazo al futuro de este proyecto.

Capítulo 2

Estado del arte

2.1. Jueces en internet

Existen muchos tipos de jueces, anteriormente hemos mencionado el Jutge, pero fuera de la UPC y fuera del ámbito académico en general hay una gran variedad. Algunos de ellos pertenecen a hacktivistas [hac, 2016] que quieren enseñar o promulgar métodos de protección y ataque a ciertos sistemas, otros se dedican única y exclusivamente a concursos, tanto para aprender [uva, 2016] como para competir [cod, 2016a] [int, 2014], algunas empresas crean juegos públicos y los mejores jugadores pueden optar a un puesto de trabajo en la empresa, es el caso de Tuenti con el Tuenti Challenge, que lo celebra una vez al año [tue, 2016]. Uno de los jueces más notables es el juez llamado codewars [cod, 2016b] el cual no es solo un juez, sino que a la vez es una red social donde entre todos hacen y corrigen ejercicios de diversos lenguajes de programación. Se trata de un juez que se construye por y para la comunidad, el profesor es a la vez el alumno que enseña a otros alumnos y toda la información posible se comparte.

El aspecto más importante del Racso como juez es que no pretende enseñar un lenguaje de programación, sino más bien conceptos de la Teoría de la Computación y por tanto, aunque juzga y se necesitan aprender ciertas tecnologías y lenguajes, la finalidad en sí misma no es la de aprender sobre los lenguajes, sino sobre conceptos; algo más abstracto que la mayoría de jueces.

2.2. Formulación del problema

Todo software necesita de un mantenimiento y el Racso no es la excepción. A veces se necesitan añadir nuevas funcionalidades, a veces se necesita mejorar el rendimiento y otras veces simplemente necesitan un lavado de cara, puesta a punto o revisión.

Después de cierto tiempo de vida útil del Racso, los profesores se han dado cuenta de que le hacían falta algunos cambios estéticos, en la parte de front-end [1], y funcionales, que le aportasen más opciones en la parte de back-end [2]. También se requería añadir un nuevo panel de administración desde el cual se pudieran acceder y ejecutar las nuevas y viejas funcionalidades de la API [3] del back-end.

Era complicado cambiar el diseño de una pantalla, por ejemplo cambiar la estructura y/o la posición de los elementos de una pantalla o cambiar el estilo de una lista de elementos. El problema residía en que el estilo y la forma estaban integradas en la lógica que procesaba la información que se pedía, es decir, allí donde se procesaba la información que queríamos observar también se procesaba la manera en la que se mostraban los datos, y este enlace hacía complejo trabajar solo con uno de los procesos. En nuestro caso si solo queríamos trabajar con el estilo debíamos mediar también con su cómputo.

Debido a la dificultad de desenlazar los dos procesos se tomó la decisión de reconstruir la API que procesa y computa, y crear una nueva capa dedicada única y exclusivamente a la presentación de los datos. Eso hará más fácil el mantenimiento de ambas capas.

2.3. Objetivos del proyecto

Como hemos indicado en el apartado anterior, hemos reconstruido back-end y front-end con el objetivo de facilitar la forma en la que se usa la aplicación, tanto a nivel de desarrollador como de usuario, que sea más usable y el código más

[1] Aquello que ve el usuario de la aplicación, administrador o no.
[2] Aquello que se encarga de computar y procesar ciertas peticiones del sistema, ya sean lanzadas por un usuario, administrador u otro sistema.
[3] Conjunto de funciones del sistema

reutilizable. Además hemos construido la nueva zona de administración y se han añadido mini-aplicaciónes de apoyo a los usuarios.

En el proceso de reconstrucción del front-end nos hemos centrado en separar toda la lógica de las pantallas en una sola capa. Llamamos a estas pantallas vistas, las vistas son fácilmente reutilizables en las distintas urls de la aplicación, son fácilmente editables y también es fácil crear nuevas vistas y añadirlas al sistema.

En la reconstrucción del back-end nos hemos centrado en que toda la antigua lógica siga funcionando de la misma forma. Es decir, que aunque hemos reconstruido la API, las postcondiciones y las precondiciones de cada llamada se cumplen de la misma forma, que como mínimo existen las mismas llamadas, aunque se llamen de forma diferente y su proceso sea distinto, pero la salida sea la misma, sin tener en cuenta su aspecto. Hemos identificado algunas funciones compuestas y las hemos sustituido por otras más atómicas y las hemos añadido a la API dotándola de un conjunto más rico y potente.

Un Administrador, desde el nuevo panel de administración, es capaz de administrar cada una de las partes del modelo del Racso. Es decir, desde aquí podemos administrar todos los elementos de la aplicación que actualmente son: alumnos, ejercicios, listas de ejercicios, exámenes y tipos de ejercicios. También es más fácil añadir nuevas entidades al modelo y que sean igual de editables que las actuales.

En cuanto al front-end hemos añadido mini-aplicaciones 3.3 que ayudan a los usuarios a resolver ejercicios y a los administradores a crear y añadir nuevos ejercicios al sistema.

Capítulo 3

Funcionalidades

Para poder definir correctamente las funcionalidades hemos hecho una lista de los usuarios con los que vamos a trabajar. Algunos de ellos serán los que utilizarán las aplicación y quienes por tanto definirán los requisitos. Otros simplemente no definirán de forma tan clara las funcionalidades, pero sí sus limitaciones.

3.1. Actores implicados

En este proyecto hemos tenido en cuenta a los diseñadores de la aplicación: director del proyecto, programador web, tester; y por otro lado tendremos a los usuarios finales: profesores que usarán la parte administrativa, alumnos de las asignaturas (alumno de examen, y alumno fuera de examen), personas externas a la universidad que quieran usar el juez.

Algunos de los actores definidos (tester, alumno, persona externa a la universidad) son solo roles que han sido interpretados por actores.

3.1.1. Director del proyecto

El actor director del proyecto ha sido Carles Creus, que ha hecho las veces de diseñador de la aplicación. Ha definido como funciona la aplicación y ha sentado la base de cambios que necesitaba el Racso para evolucionar. Además ha supervisado el avance del proyecto y el cumplimiento de las bases definidas.

3.1.2. Programador

El programador web ha sido Jaime Pascual Torres. Se ha encargado de la tarea de reconstrucción del Racso en base a las necesidades definidas por el Director/Diseñador principal. Junto con el director ha tomado la decisión de elegir la o las tecnologías que más se adaptan a las necesidades.

3.1.3. Diseñador

El rol de Diseñador ha sido ejercido por el programador Jaime Pascual Torres. Ha utilizado algunos frameworks para diseñar el front-end de la aplicación. Se ha basado en dos librerias: Bootstrap[1] y Materialize[2]

3.1.4. Tester

El rol de tester lo han tomado algunos alumnos, ex alumnos de la asignatura y profesores adjuntos para testear y probar la aplicación. Además, el programador ha seguido la práctica de la integración continua, la cual ha proveído al proyecto de código extra enfocado a probar y verificar el correcto funcionamiento de la aplicación Racso. Entre los cuales destacamos los tests funcionales, que recrean el comportamiento de un usuario en la aplicación y revisan los resultados de este comportamiento.

[1] http://getbootstrap.com/
[2] http://materializecss.com/

3.1.5. Administrador

El administrador también es un rol que han tomado los profesores de la asignatura de Teoría de la computación. Carles Creus y el resto de miembros que forman el profesorado han sido los que han jugado este papel. Estos poseen un acceso especial desde el cual se puede modificar la información vital de la aplicación, alumnos, ejercicios, listas de ejercicios, exámenes y tipos de ejercicios.

3.1.6. Alumnos

Este rol está dividido en dos partes y será importante distinguirlas ya que la aplicación no se comporta de igual forma en modo examen que fuera de él. Aunque será interpretado por las mismas personas en ambos casos (programador, director, otros profesores y ex alumnos) estos tendrán que tener en cuenta esta diferencia entre modos.

3.1.7. Personas externas a la universidad

Este rol es también necesario e importante, se busca abrir el Racso a un mercado mayor, no solo entre los estudiantes de la universidad, si no también a interesados en la materia de fuera del círculo de la FIB. Hemos tenido en cuenta y dado oportunidades a un usuario que no está acostumbrado al funcionamiento ni a las metodologías de la FIB. Ya que no existen actores que hayan interpretado este papel y que estén vinculados al proyecto, han sido los otros actores quienes han desempeñado este papel. Esto lo convierte sin duda en el más difícil de todos ya que ninguno de ellos es externo a la universidad.

3.2. Lista de funcionalidades

Existen dos tipos de funcionalidades, las funcionalidades públicas, a las que cualquier usuario, registrado o no, puede acceder, y las funcionalidades privadas, a las que solo los administradores pueden acceder y desde las cuales pueden administrar la aplicación.

3.2.1. Funcionalidades públicas

- Ejercicios
  - Acceso a los ejercicios públicos
    - Cualquier usuario, identificado o no tiene que poder acceder a un ejercicio público, poder entregarlo y ver su corrección.
  - Listas de ejercicios
    - Cualquier usuario, identificado o no, tiene que poder ser capaz de ver las listas públicas de ejercicios.
  - Listado de entregas de un ejercicio
    - Los usuarios identificados que han entregado los ejercicios mientras estaban identificados tienen que poder ver todas sus entregas anteriores sobre ese ejercicio.
- Exámenes
  - Ver un examen
    - Los usuarios identificados a los que se les ha puesto un examen[3] solo tienen acceso a ese examen, de modo que no pueden hacer ninguna de las otras acciones durante ese tiempo.
  - Ver los ejercicios de un examen
    - Un usuario identificado al que se le ha puesto un examen tiene que poder ver los ejercicios de ese examen.
  - Entregar los ejercicios de un examen
    - Un usuario identificado al que se le ha puesto un examen tiene que poder entregar los ejercicios del examen.
  - Ver la nota
    - Un usuario identificado al que se le ha puesto un examen tiene que poder ver su nota, tanto la total como la de cada ejercicio.
  - Bloqueo previo y posterior

[3]Es una función privada que puede hacer un administrador.

  - El Racso se bloquea para los usuarios identificados que tienen un examen un cierto tiempo antes del examen y un cierto tiempo después del examen.

- Bloqueo en las aulas de examen
  - Los usuarios no identificados o los usuarios identificados pero que no tienen el examen al que se le ha asociado el aula desde donde acceden, no pueden acceder a ninguna de las funcionalidades durante el tiempo de examen.

3.2.2. Funcionalidades privadas

Una de las funciones privadas más importantes es poder editar todos los atributos de cada entidad del modelo así como crear nuevas instancias de las entidades del modelo o eliminar las instancias.

Casi todas las entidades poseen nombre y descripción. Algunas entidades como el Ejercicio poseen campos especiales como los flags de compilación, el tiempo máximo de ejecución o la memoria máxima de ejecución. En el caso de las listas, como por ejemplo Listas de ejercicios o Exámenes, hay un campo en el que se indican múltiples ids para crear el enlace entre los ejercicios o el enlace entre el examen y sus ejercicios respectivamente. Para esto hemos dotado a las vistas de pequeños botones y selectores.

Los administradores también pueden revisar todas las entregas de todos los ejercicios. Durante el examen ven las notas que ha logrado cada alumno hasta el momento. También tienen a su disposición una lista de IP-Alumno que indica, para cada alumno, la IP desde la que está conectado. Esta IP se guarda durante la primera sesión del Alumno en el examen y no puede cambiar durante ese examen, para evitar que los alumnos se copien las soluciones de los ejercicios si conocen las contraseñas de sus compañeros. Este enlace solo puede ser eliminado por el administrador.

Se ha dotado de una funcionalidad privada a los administradores que les ayuda a crear y editar ejercicios. Esta funcionalidad está explicada en detalle en el apartado 3.3

3.3. Mini-aplicaciones de apoyo a los usuarios

Se ha preparado el sistema para que puedan añadirse fácilmente mini-aplicaciones web asociadas a cada tipo de ejercicio. El objetivo de estas mini-aplicaciones es dotar al sistema de sistemas de ayuda, tanto para el administrador en sus tareas de preparación de ejercicios, como a los alumnos mientras tratan de resolver ejercicios. En particular, se ha detectado que los alumnos tienen en ocasiones problemas para comprender algunos veredictos cuando realizan un envío erróneo, y mediante estas mini-aplicaciones se pretende ofrecer la posibilidad de que, interactivamente, un alumno pueda obtener mayor información sobre el error que hay en su propuesta de solución.

Se ha tomado la decisión de diseñar estas funcionalidades como mini-aplicaciones independientes de los jueces C++ por dos motivos. En primer lugar, las mini-aplicaciones se ejecutarán en el ordenador del cliente, evitando así añadir carga computacional al servidor. En segundo lugar, se evita tener que modificar y ampliar el código C++ de los jueces, que en su estado actual ya resulta una implementación sofisticada y crítica en tiempo de ejecución.

3.3.1. Mini-aplicación ya disponible

El Racso ya contaba con una mini-aplicación que permite calcular accesibilidad de palabras, es decir, una mini-aplicación que recibe dos palabras u y v junto con un sistema de reescritura R (conjunto de reglas de la forma $l \to r$, que indica que la subpalabra l puede ser reescrita y convertirse en la subpalabra r) y trata de averiguar si u puede reescribirse a v aplicando las reglas de R en cualquier orden y tantas veces como sea necesario. Por supuesto, accesibilidad de palabras es un conocido problema indecidibile [GOEDEL, 2006], y la mini-aplicación símplemente trata de detectar si la palabra v es accesible desde u con un cierto número de pasos prefijado. Esta restricción extra en el número de pasos permite que el problema se vuelva computable, pero más simplificaciones son necesarias para lograr que la mini-aplicación resulte eficiente, como por ejemplo acotar más aún el espacio de búsqueda.

El objetivo de esta aplicación es ayudar a entender veredictos en los ejercicios de reducciones de accesibilidad de palabras. Por ejemplo, un veredicto para un

cierto ejercicio podría rechazar un envío aduciendo que con el sistema de reglas:

$$\begin{aligned} a &\to cc \\ b &\to cc \\ c &\to cc \\ acc &\to cc \\ bcc &\to cc \\ ccc &\to cc \\ cc &\to caac \\ cac &\to acac \\ cac &\to bcac \\ cac &\to ccac \\ cac &\to c \end{aligned}$$

se puede reescribir la palabra a a la palabra c. Esto puede confundir a un usuario, pues no resulta evidente qué pasos de reescritura serían necesarios, más aún si el usuario está en un contexto de presión como un examen. La mini-aplicación permite calcular la derivación:

$$a \to cc \to caac \to cccac \to ccac \to cacac \to cac \to c$$

3.3.2. Mini-aplicación añadida

Los ejercicios sobre gramáticas libres de contexto (CFG, por sus siglas en inglés [Nelson, 2016]) suelen presentar dificultades a los alumnos de Teoría de la Computación. Las gramáticas son un sistema generativo para lenguajes[4], es decir, describen un lenguaje a partir de un conjunto de reglas recursivas que definen la forma en que pueden generarse todas las palabras del lenguaje. El ejemplo típico de gramática es

$$\begin{aligned} S &\to aSb \\ S &\to \varepsilon \end{aligned}$$

que especifica que la variable S puede generar la palabra aSb (en la que recursivamente se debería reescribir la nueva ocurrencia de S) o la palabra vacía (denotada por ε). El lenguaje generado por esta CFG ejemplo es el conjunto de palabras cuya primera mitad son todo a's y cuya segunda mitad son todo b's: la palabra vacía, ab, $aabb$, $aaabbb$, $aaaabbbb$, etc.

[4]En este contexto, un lenguaje es un conjunto de palabras.

Los ejercicios del Racso sobre CFG piden que se definan gramáticas que generan lenguajes específicos y, en algunos casos, se exige además que la gramática sea inambigua (es ambigua si la gramática puede generar alguna palabra del lenguaje de dos formas distintas). Por tanto, los veredictos del juez pueden rechazar un envío cuando se detecta la existencia de una palabra que cumple alguna de las siguientes propiedades:

- no es generada por la gramática y debería serlo,
- es generada por la gramática y no debería serlo,
- es generada de forma ambigua por la gramática cuando se exigen gramáticas inambiguas.

El veredicto incluye la palabra en la que se ha detectado el problema, pero en ocasiones los alumnos pueden tener dificultades para averiguar la forma en la que la gramática que enviaron genera la palabra errónea, en especial si el motivo era que la generación es ambigua. Por esta razón, se ha decidido añadir una mini-aplicación con la que, dada una gramática y una palabra, el usuario pueda obtener una derivación de cómo la gramática genera esa palabra.

Capítulo 4

Diseño

La aplicación consta de diversas partes, la parte principal está programada en C++ y se encarga de las correcciones de los ejercicios, después viene la parte web que es la que hemos reconstruido. Durante la reconstrucción y sobre todo en la parte de diseño hemos tenido en cuenta el posible crecimiento de la aplicación, tanto en cantidad de usuarios como en servicios. Por ello la hemos diseñado con el mayor desacoplamiento posible, haciendo así que todas las partes sean independientes y se puedan comunicar unas con otras. De este modo, la aplicación web funciona como una API contra la que el resto de servicios se han de identificar para mandarle peticiones y que esta les conteste. Esto le confiere al Racso la posibilidad de crecer fácilmente.

4.1. Patrón Modelo Vista Controlador

Como hemos dicho anteriormente uno de los principales motivos de la reconstrucción era tratar de separar la capa de vistas de la capa de lógica, logrando así que sea mucho más cómodo y fácil implementar nuevas funcionalidades en el Racso. Esto lo hemos conseguido aplicando el Patrón Modelo Vista Controlador [mvc, 2014], el que desacopla el modelo del sistema, las clases y su estructura (Modelo), del de las vistas que muestran los datos (Vista) del código que computa y procesa las diferentes peticiones (Controlador). Este código ha quedado encapsulado dentro de Controladores.

4.2. Proceso de autentificación

Uno de los procesos más importantes a tener en cuenta en el diseño es el proceso de autentificación. En una primera visión podemos pensar que es un proceso trivial en el que un usuario nos proporciona un nombre y una contraseña que validamos contra la base de datos y así es como identificamos y autentificamos al usuario. Sin embargo en el Racso hemos tenido en cuenta una serie de factores extra: además de identificar a los usuarios registrados, mantenemos controladas sus sesiones para poder enviarles eventos en cualquier momento.

Para aumentar la seguridad del Racso hemos creado dos tipos de usuarios, los Users y los Admins. De forma que al no ser el mismo tipo de usuarios no se validan contra la misma entidad del modelo. Esto lo hemos logrado gracias a una característica de Laravel que nos permite desarrollar procesos específicos para ciertos tipos de rutas [1], así, todas las rutas que empiezan o contienen la palabra admin son contrastadas en la base de datos con un código y las rutas que no lo contiene con otro.

En general el proceso de autentificación funciona por capas, para lograr acceder a la capa de datos debes pasar por todas las capas anteriores. A continuación listaremos las distintas capas que posee el sistema y su finalidad:

- Capa web

  Esta capa se encarga de comprobar que la sesión del usuario es la correcta.

- Capa autentificación usuario

  Esta capa se encarga de comprobar la autentificación de los usuarios normales.

- Capa autentificación administrador

  Esta capa se encarga de comprobar la autentificación de los usuarios administradores.

- Capa de autentificación de examen

  Esta capa se encarga de verificar y hacer ciertas comprobaciones en el modo de examen.

En el diagrama de capas de autentificación 4.1 podemos comprobar cual será la estructura de estas capas.

4.2.1. Autentificación en base al examen

Esta capa, como su nombre indica, se encarga del proceso de autentificación durante un examen. Procesa ciertos parámetros como la localización IP, la asociación de la IP a un examen, el momento en el que se accede y si el usuario que accede debería estar o no haciendo un examen y cuál. Estos parámetros están asociados y el Racso debe ponerse en modo examen para advertir a un usuario que está en otra aula que debe hacer el examen o para advertir a un usuario que está en un aula que en esa aula se va a proceder a hacer un examen. En tiempo de examen solo se puede acceder a los ejercicios del examen si y solo si tu usuario tiene los permisos indicados para ello por el administrador.

4.3. Proceso de entrega de un ejercicio

En la reconstrucción se ha conseguido que este proceso sea más dinámico y ágil. Antes el Racso, después de que un alumno enviase un ejercicio, hacía hasta 5 peticiones con un intervalo de tiempo de 2 segundos para comprobar que la solución del ejercicio está lista. Hemos rediseñado este proceso para que sea a la inversa, que el servidor mande un evento al cliente que le indique que su ejercicio ha sido corregido y además le mande la corrección. Para ello necesitamos que el usuario esté identificado en todo momento y usaremos la librería Socketio [2] de JavaScript para crear un socket entre el cliente y el servidor. A simple vista el alumno envía una petición de juicio, el servidor la procesa y cuando la ha procesado le contesta, pero el proceso es ligeramente diferente, ya que la corrección no la hace el servidor Laravel si no la parte en C++, el control de los eventos para con el cliente los hace el servidor Node y el servidor Laravel se encarga de comunicar entre el cliente web, el cliente en C++ y el servidor Node.

En la figura 4.2 se puede ver cuales son los pasos a seguir en el proceso de

[1]Un acceso a un recurso.
[2]http://socket.io/

entrega de un ejercicio.

4.4. Diseño de la web

La interfaz de cara al usuario apenas ha cambiado pero se han pulido algunas partes o zonas de cada vista. Hemos facilitado a los usuarios el acceso a una lista de ejercicios en base al ejercicio en el que se encuentran, es decir, cada ejercicio público pertenece a una lista pública y esta lista es fácilmente accesible desde la pantalla donde se muestra el ejercicio. La solución que hemos llevado a cabo para que sea así es la de colocar un breadcrumb [3] que nos ayudará a la navegabilidad de la web en todo momento. Este breadcrumb se usará en todas las vistas. También hemos definido un poco más la zona de output de un ejercicio y le hemos aportado más visibilidad, así sabemos rápidamente si una solución es correcta o incorrecta. Cuando el administrador-profesor activa el modo examen, y gracias a los eventos proporcionados por la librería socketio, podremos enviarle un mensaje al Alumno para advertirle de que ya puede empezar el examen o para advertirle de que el tiempo de examen se está acabando.

En cuanto al panel de administración, que es donde más información hemos de controlar, hay 3 tipos de pantallas: el dashboard, el listado de elementos de una entidad (ejercicios, usuarios, listas de ejercicios, exámenes, kinds y aulas) y la pantalla de edición de cada una de estas.

Las pantallas de listados las hemos resuelto con usando DataTables [4] que nos permite crear una tabla con toda la información. Se pueden editar las columnas de esta tabla en todo momento, de modo que comenzaremos cargando la tabla con los campos o columnas más importantes y daremos la opción al administrador para que decida qué campos quiere ver y cuáles no. También podrá ordenar fácilmente los elementos por cada uno de los campos y buscar por palabras clave. Tiene una serie de botones que le permiten exportar los datos a distintos formatos tales como pdf, excel, imprimir, copiar y otra serie de botones que le permiten acceder fácilmente a la edición o clonación de una instancia. En la figura 4.3 se muestra el aspecto de una de las tablas y en la figura 4.4 la selección de columnas.

[3] Cadena o rastro que te permite volver a páginas anteriores.
[4] https://datatables.net/

La pantalla de edición de los modelos es una pantalla genérica que muestra un formulario. Dependiendo del modelo, este formulario es más o menos complejo, llegando a poseer pequeños editores que generalmente servirán para editar texto LaTeX para títulos y descripciones de las entidades. Estos van acompañados de pequeños outputs donde se verá el ejemplo de salida en HTML. En la figura 4.5 tenemos un ejemplo de ello.

Todos estos elementos son reutilizados para la vista de examen de modo que el administrador verá una tabla con una columna por ejercicio y las notas que tiene cada alumno hasta ese mismo instante. Verá también otra tabla con la relación entre alumno y la IP desde la que está conectado, de modo que pueda eliminar esa conexión por si un PC se estropea en tiempo de examen. Este proceso se explica en el apartado 4.2.1.

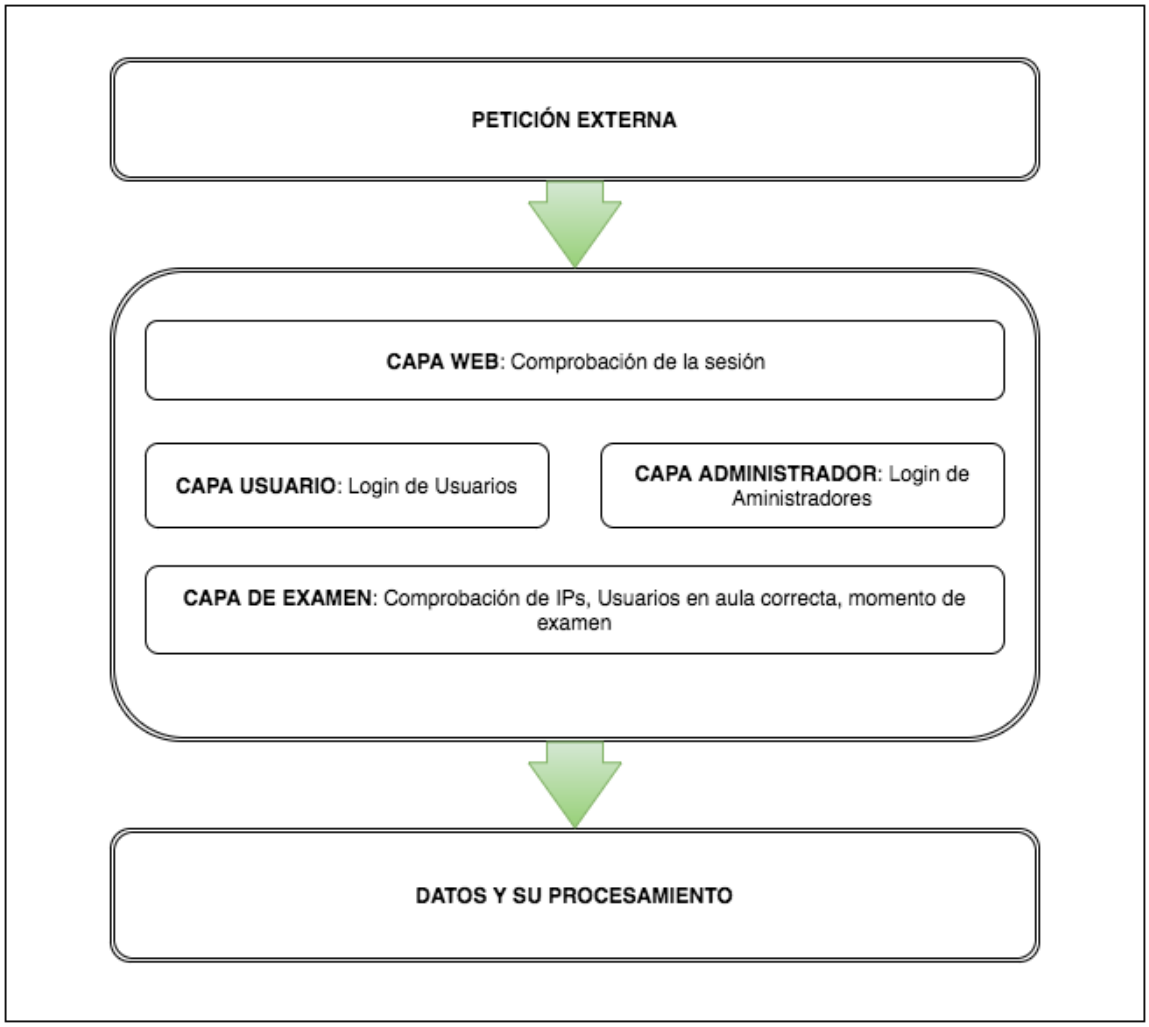

Figura: 4.1: Diagrama de capas de autentificación

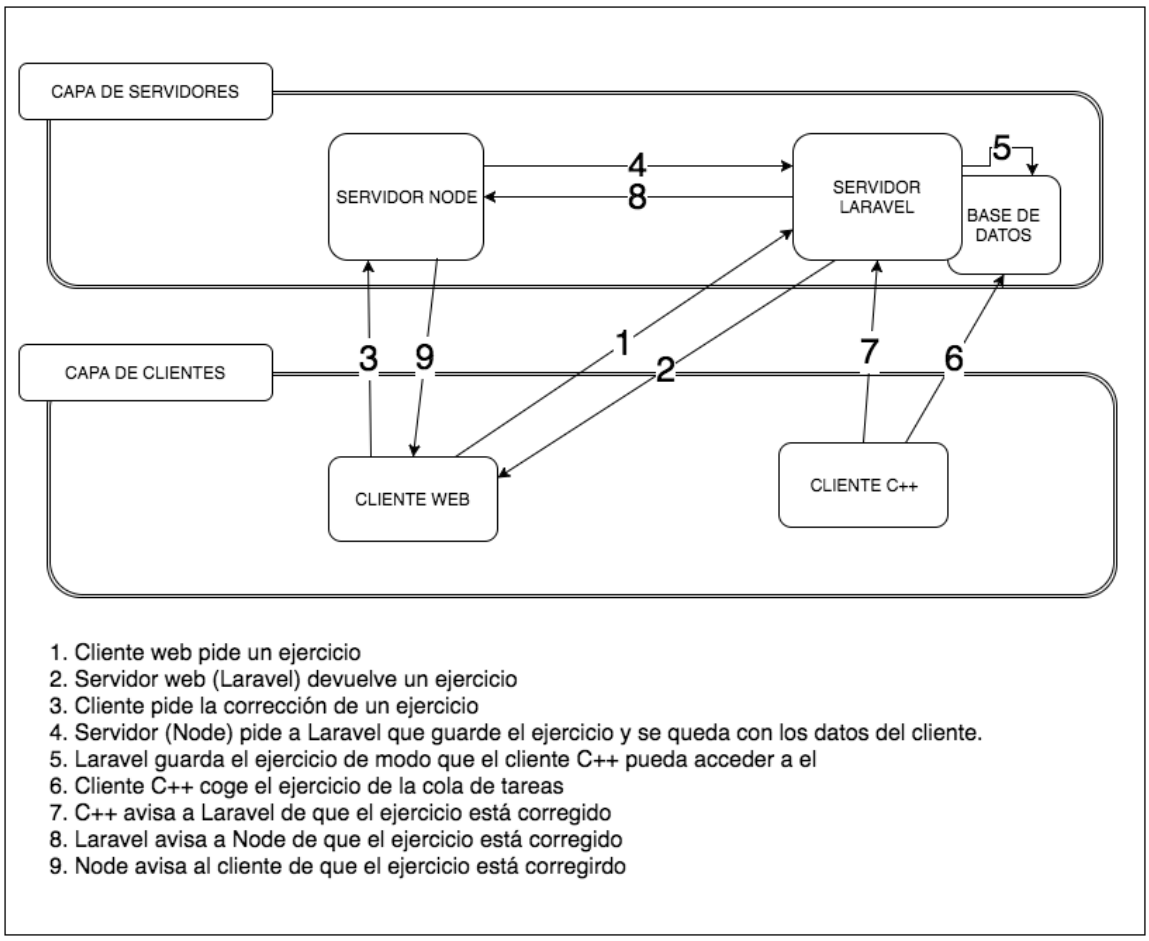

Figura: 4.2: Diagrama de la corrección de un ejercicio.

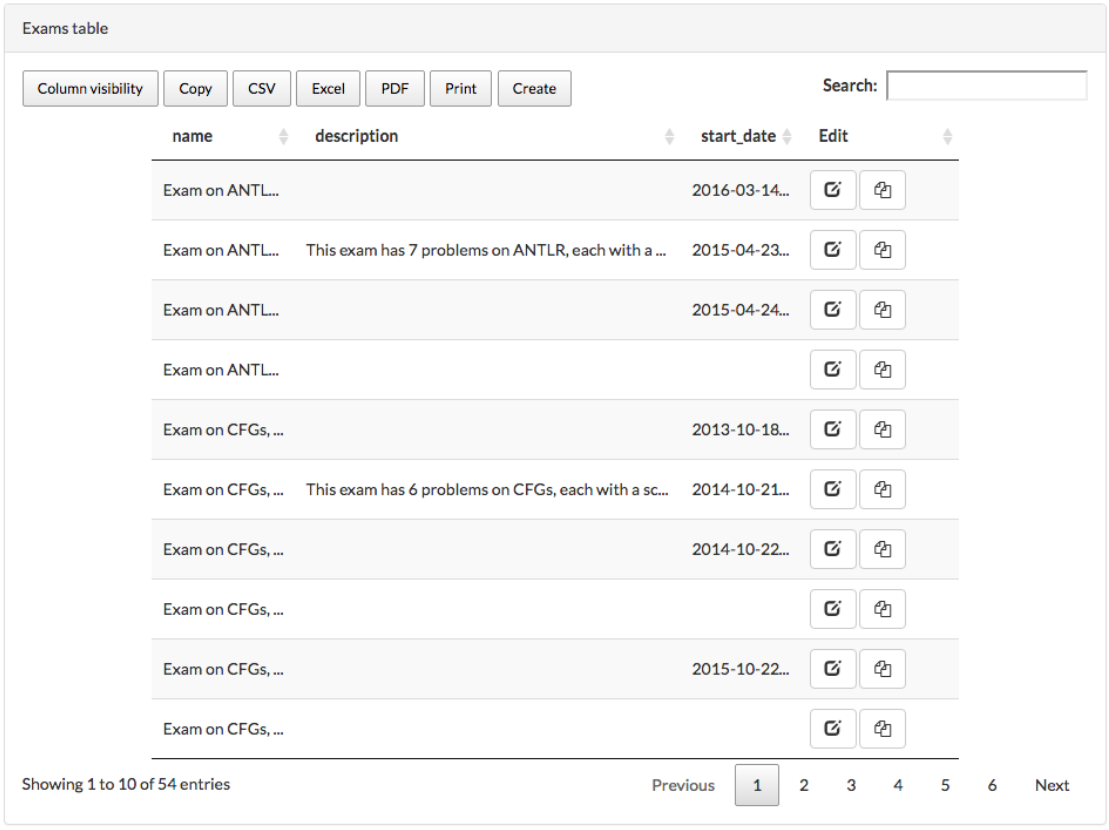

Figura: 4.3: Tabla de exámenes

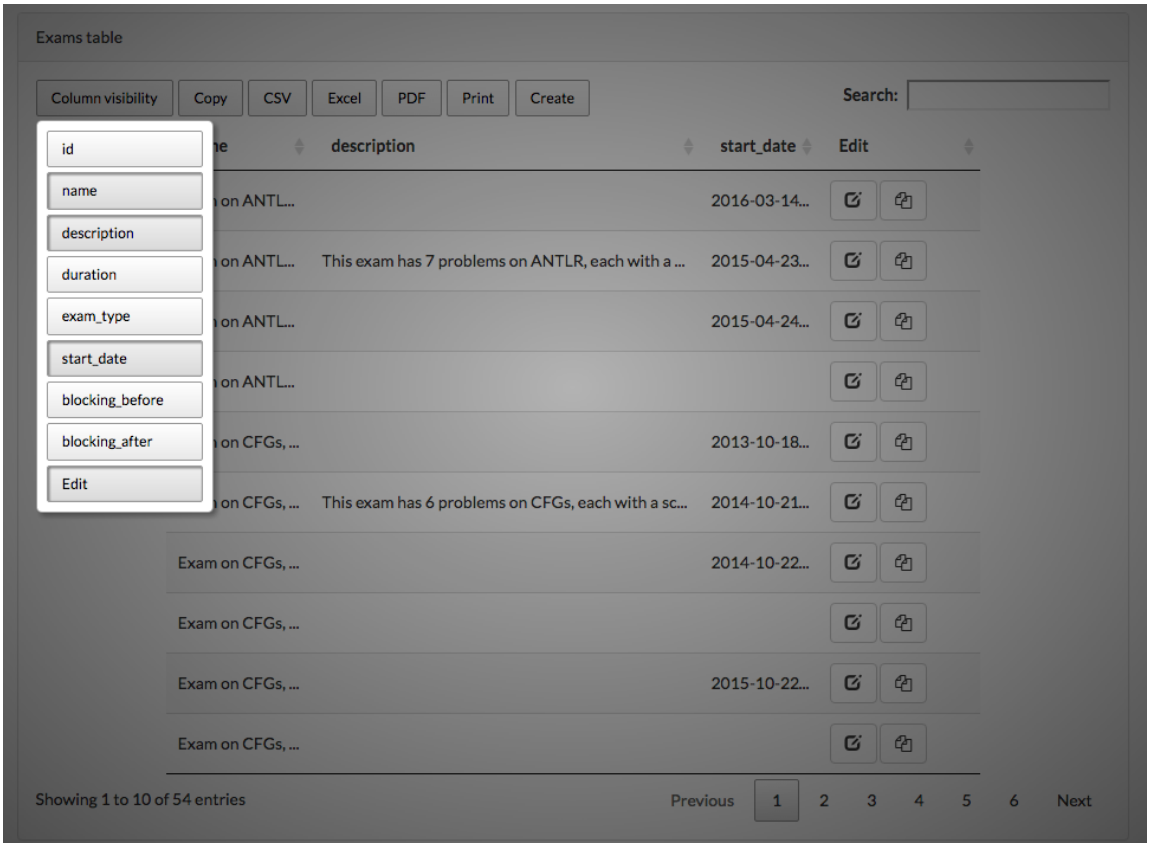

Figura: 4.4: Edición de columnas en la tabla de exámenes.

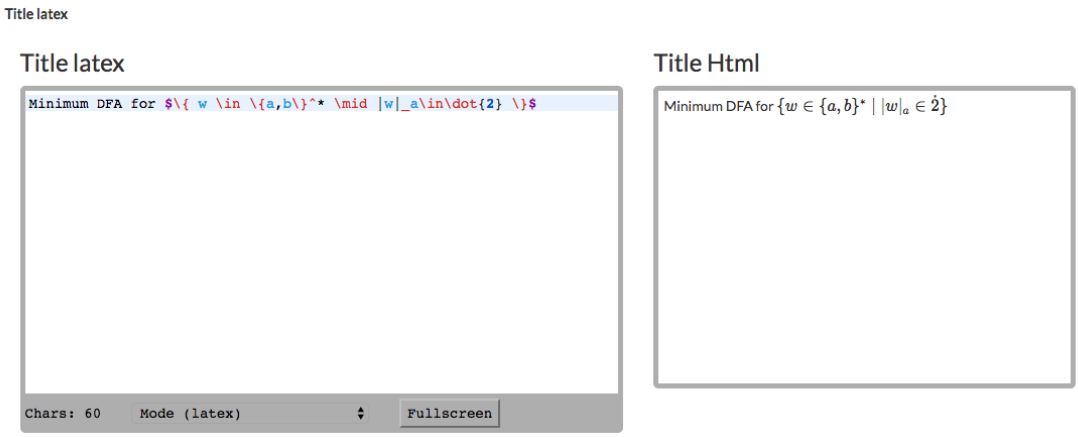

Figura: 4.5: Editor LaTeX con output HTML.

Capítulo 5

Implementación

5.1. Tecnologías con las que vamos a trabajar

Comenzamos el proyecto seleccionando las tecnologías que hemos empleado para implementarlo. La decisión sobre cuáles tecnologías utilizar para este proyecto ha sido tomada en base a una serie de factores que presentaremos a continuación en orden de prioridad:

- Contexto de utilización: La tecnología se usa habitualmente en contextos parecidos al de nuestra aplicación.
- Conocimiento previo por parte del programador.
- Estado de la comunidad, facilidad para encontrar información y utilidad de la misma.
- Ganas de aprender por parte del programador.

En anterior Racso utilizaba las tecnologías apache [1], PHP, C++ y JavaScript para dar el servicio. Apache, PHP y JavaScript se encargaban de la parte pública de la web y C++ de la parte privada. La parte que hemos reconstruido es

[1] http://www.apache.org/

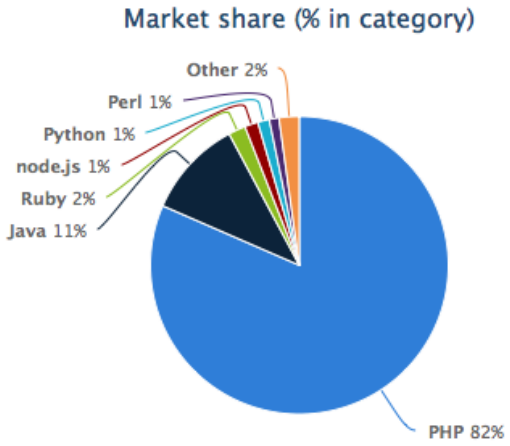

Figura: 5.1: Proporción de proyectos web por lenguaje de programación. [wap, 2016]

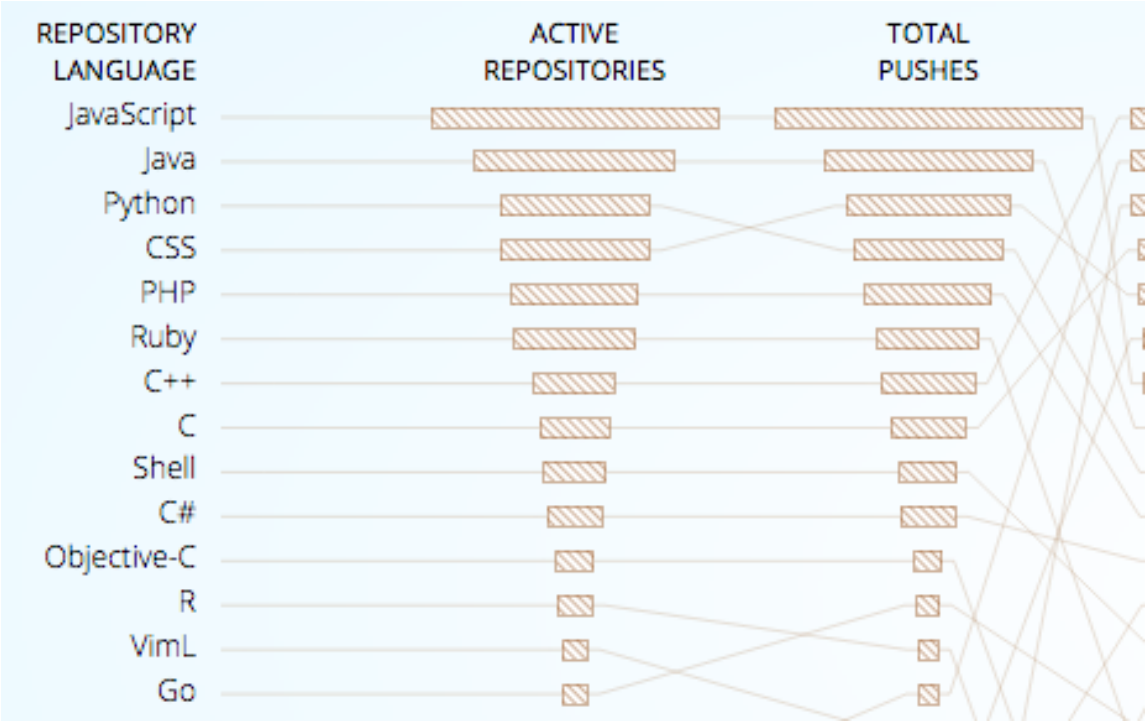

Figura: 5.2: Top 14 Lenguajes con proyectos mas activos en GitHub.

la parte pública que procesa y muestra el resultado de las peticiones que le llegan a través de la red. De esta parte se encarga más concretamente PHP y JavaScript, que, unidos, computan toda la lógica de back-end y front-end. PHP ejecutándose en la parte de servidor, y no en la parte de cliente, y JavaScript ejecutándose en la parte de cliente para hacer que las pantallas sean un poco más dinámicas. Este último podría también encargarse de algunas partes de back-end en el servidor, pero por ahora no se da el caso.

Como podemos ver en la imagen 5.1 alrededor del 80 % de las webs en internet utilizan PHP y solo el 1 % Node, que es la forma en la cual JavaScript se puede ejecutar en el lado del servidor. Por otro lado en la imagen 5.2 podemos ver que JavaScript es el lenguaje con más actividad de la comunidad GitHub y PHP está el quinto lugar en esta misma comunidad. En esta y otras comunidades parecidas, como pueden serlo Stack Overflow y Google Groups, existe mucha actividad al respecto de estos dos lenguajes, por lo tanto fue fácil encontrar la información que necesitábamos. En general son dos lenguajes con los que se trabaja mucho para desarrollar una aplicación web. Se trata de dos lenguajes de scripting que ayudan a mantener un desarrollo ágil y continuo.

Ha sido una buena opción reconstruir el Racso con las mismas tecnologías con las que fue construido la primera vez. El programador había utilizado previamente estos dos lenguajes, se sentía cómodo con ellos y quería aumentar su conocimiento de estos mismos. PHP y JavaScript cumplieron todos los requisitos y por tanto fueron los lenguajes con los cuales se implementó el nuevo Racso.

5.2. ¿Por que un framework?

El anterior Racso fue desarrollado sin la utilización de ningún tipo de framework [2]. Dado el tamaño y la complejidad del proyecto, creímos que esta vez lo mejor sería aprovecharse de alguna de las herramientas especificadas en el párrafo siguiente. De este modo podríamos centrarnos en el desarrollo de las funcionalidades y no tanto en las tecnologías que usaríamos. Gracias a ello las tareas están más centradas en procesos específicos, también podemos aprovechar las herramientas que nos da el framework para evitar perder tiempo en trabajos triviales y repetitivos [Symfony, 2016] como la autentificación de un usuario y también para automatizar

[2]Una infraestructura o marco sobre la que desarrollar un software.

ciertos procesos como mantener el control de las migraciones de la base de datos.

El Framework que decidimos utilizar para este proyecto sospesando sus virtudes y carencias es Laravel, que lleva creciendo en popularidad y utilización desde los últimos tres años. Este reaprovecha la estructura y filosofía de los componentes del framework Symfony con un pequeño añadido: se centra en mantener las cosas simples. Nos planteamos hacerlo con el Symfony 2, que es uno de los frameworks más famosos dentro de la comunidad de PHP, ya que ha asentado muchas de las bases que a día de hoy son estándares en casi cualquier proyecto de PHP, pero por desgracia es costoso de lograr ya que al ser un framework que viene completamente en blanco, requiere de un gran conocimiento previo de los componentes para poder sacarle el máximo partido. Laravel viene de serie con un conjunto de componentes previamente configurados y probados y ha reimplementado algunas soluciones más sencillas de las que hablaré en el párrafo siguiente.

Como ya he dicho, Laravel tiene virtudes y desventajas y es que su simplicidad que a simple vista podría ser una gran virtud, puede convertirse en un arma de doble filo. Más sencillo a veces implica menos potente o menos posibilidades de configuración y personalización. Por suerte hemos solventado ese problema ya que Laravel nos permite introducirnos en su workflow y cambiarlo a nuestro gusto para añadir fácilmente configuraciones que de serie no vienen. Además la configuración por defecto se adapta lo suficiente a nuestras necesidades. En general el tipo de acciones que hemos llevado a cabo en este proyecto son complejas peticiones a la base de datos, el control de acceso y el procesamiento de estos datos en base a este controlado acceso. Laravel viene de serie con Eloquent, un potente ORM [3] que nos ha ayudado a mapear la base de datos en código PHP de forma casi instantánea y segura. Así podremos controlar fácilmente el acceso a la base de datos, sin preocuparnos de la conexión, o de si estamos escribiendo de forma correcta en la sintaxis de MySQL o sin siquiera preocuparnos del tipo de base de datos que hay detrás. Con Eloquent podemos trabajar libremente sobre la base de datos sin preocuparnos de cuál es la implementación de esta misma. Por otro lado el IoC[4] container nos ha ayudado a manejar las dependencias de código entre las distintas partes y procesos de la aplicación [Documentation, 2014], algo que comúnmente nos distrae mientras programamos aquello que queremos lograr, sobre todo en complejos procesos con múltiples capas. Por último los patrones adoptados por Laravel como MVC y Factory Pattern nos han ayudado a mantener una estructura fuerte y segura en la

[3]Herramienta para abstraer la forma en la que se utiliza la base de datos.
[4]Inverse of control.

aplicación [pat, 2014].

5.2.1. Librerías de JavaScript

Con JavaScript podríamos haber hecho lo mismo que hicimos con PHP y buscar un framework o librería que nos ayudase a mantener el código organizado y a tener una base de funciones y/o herramientas con las cuales trabajar en el front-end en el lado del cliente. Pero en este caso hemos optado por no usar ningún framework ya que el nivel de uso del JavaScript en este proyecto es específico para ciertos puntos muy detallados. Lo que sí hicimos previamente a empezar a programar es identificar algunas librerías que creíamos necesarias para el proyecto. Aquí está la lista de librerías que identificamos en su momento junto con la lista de librerías que hemos usado finalmente.

- Node: Es un motor JavaScript que nos ha ayudado a correr las librerías que necesitamos para procesar algunas de las peticiones que recibirá el servidor.
- Socketio: Se trata de una librería JavaScript que usamos para mantener una conexión abierta entre el cliente y el servidor, de modo que el servidor puede enviarle eventos importantes como la corrección de un ejercicio [5] al usuario.
- Ioredis: Se trata de un cliente JavaScript que añadimos a Node para que entre él y Laravel puedan compartir mensajes mediante un servicio de base de datos en memoria llamado Redis. De esta forma Laravel y Node podrán compartir mensajes y enviarse eventos.
- Gulp: Es una librería JavaScript que hemos usado para definir una serie de tareas y un conjunto de reglas. Esta librería la usamos sobre todo para el desarrollo, la automatización de los tests y la compilación del Sass [6].
- DataTables: Librería que nos permite crear una tabla con la que podemos interactuar de muchas formas. Podemos ordenar los elementos de la tabla según sus atributos en orden ascendente o descendente. También podemos buscar elementos en la tabla mediante palabras clave.

[5] Algo que durante el proyecto hemos denominado como .[el] veredicto del juicio".
[6] El lenguaje de estilos que utilizamos en el proyecto. http://sass-lang.com/

5.3. Plataformas de despliegue y desarrollo

En esta sección definiremos las características de las máquinas de desarrollo y producción, así como el control de las mismas, su aprovisionamiento [7] y las actualizaciones de sistemas controladas. La idea básica es que la máquina donde se desarrolla es una copia de la máquina donde va a estar el producto final. De esta forma se evitan comportamientos inesperados y fallos en el momento de publicar la aplicación, reduciendo así el impacto del despliegue.

5.3.1. Máquina de desarrollo y despliegue en producción

El proceso de desarrollo ha transcurrido en una máquina virtual, de forma que todo el código está autocontenido en ella. Con esto logramos que la máquina se ejecute de la misma forma en el PC de cualquier desarrollador. Para ello hemos usado Vagrant [8] que nos ha permitido definir una máquina virtual ligera y altamente reproducible que se aprovisionará siempre de la misma forma, con las misma versión de programas, evitando así problemas entre máquinas y versiones de programas y sistemas. Vagrant requiere de una instalación previa de VirtualBox [9]. Hemos usado Vagrant para la publicación del software de forma que, indicándole un servidor remoto, este se encarga de copiar toda la configuración y archivos de la máquina virtual y la de desarrollo, en la máquina de producción. Una vez hecho este proceso solo hemos tenido que aprovisionar la máquina.

5.3.2. Aprovisionamiento de las máquinas

Para aprovisionar las máquinas, tanto la de desarrollo como de producción, hemos utilizado una herramienta que instala y configura las dependencias automáticamente. Con dependencias nos referimos a los programas, lenguajes, librerías, compilaciones, todo tipo de proceso previo que requiera el proyecto para correr de forma normal. Previamente hemos tenido que definir estas dependencias en scripts y algunos

[7] Proceso por el cual una máquina o un sistema se dota de los programas que requiere para su correcto funcionamiento.

[8] Entornos de desarrollo virtuales, ligeros y re-producibles https://www.vagrantup.com/

[9] Aplicación para la virtualización de sistemas https://www.virtualbox.org/

archivos YAML[10] de configuración. El encargado de todo este proceso será Ansible [11], una herramienta de automatización capaz de controlar múltiples máquinas.

5.4. Automatización de las pruebas

Ha sido importante nutrir al proyecto de una buena cantidad de tests que nos indiquen que todo funciona correctamente y que no rompemos nada al añadir nuevas funcionalidades. En este apartado hablaremos de los tipos de test que hemos utilizado y cómo hemos aplicado estas tecnologías.

5.5. Implementación mini-aplicaciones

5.5.1. Tipos de tests

Existen muchos tipos de tests, pero en este proyecto nos hemos centrado en dos tipos, los test unitarios y los tests funcionales. Hemos usado los test unitarios para cerciorarnos de que ciertas partes del código funcionan correctamente y los tests funcionales para controlar el flujo de la aplicación.

Los elementos susceptibles de ser controlados por tests unitarios son las clases del modelo y los controladores, cada uno por separado. Se han testeadao los elementos que poseen funcionalidades complejas, distintas a los típicos getters y setters, los cuales no testearemos ya que esto depende de Eloquent y este ya viene testeado de serie. Todo ello para comprobar que la clase funciona correctamente y que no hay comportamientos inesperados que puedan romper la aplicación.

Por otro lado tenemos los tests funcionales. Se trata de un test de flujo en la aplicación que imita el comportamiento de un usuario y comprueba que el resultado obtenido es el esperado.

Para todo ello hemos usado dos librerías, PHPUnit [12], para los tests unita-

[10]Un formato que nos propone un estándar para archivos de configuración. http://yaml.org/
[11]Automatización de levantado y aprovisionamiento de máquinas. https://www.ansible.com/
[12]https://phpunit.de/

rios, y Selenium[13], para los test funcionales. Requerimos de Selenium ya que parte de la funcionalidad transcurre en JavaScript y PHPUnit no es capaz de ejecutarlo.

5.5.2. Idea del algoritmo

Nos basamos en el algoritmo de Cocke-Younger-Kasami (CYK, [Har-Peled and Parthasarathy,
La idea de este algoritmo es simple: dada una CFG G y una palabra w, se define $gen(X, u)$ para cada variable X de G y cada subpalabra u de w tal que $gen(X, u)$ es cierto si X puede generar u, y falso en caso contrario. Con esta información se puede calcular fácilmente si la gramática genera la palabra w, pues solo es necesario mirar si $gen(S, w)$ es cierto para la variable inicial S de G. Además, si todas las reglas de la gramática G son o bien de la forma $X \to a$ o bien de la forma $X \to YZ$, siendo a un símbolo del alfabeto y X, Y, Z variables, entonces $gen(X, u)$ se puede definirse de forma recursiva: el caso base es cuando u tiene tamaño 1 y el valor de $gen(X, u)$ símplemente depende de si existe o no una regla de la forma $X \to u$; el caso recursivo es cuando u tiene tamaño mayor que 1 y entonces el valor de $gen(X, u)$ depende del valor de $gen(Y, u_1)$ y $gen(Z, u_2)$ para todas las variables Y, Z tal que existe una regla $X \to YZ$ y para toda u_1, u_2 tal que su concatenación da como resultado u. El beneficio de esta formulación recursiva es que puede implementarse de forma eficiente usando programación dinámica, obteniendo un algoritmo de coste $\mathcal{O}(|G| \cdot |w|^3)$.

Hemos tenido que adaptar el algoritmo CYK a nuestros fines, pues las gramáticas de los alumnos pueden tener reglas con formas más complejas que $X \to a$ o $X \to YZ$, y además nuestro objetivo no es solo averiguar si la gramática genera una palabra sino obtener la derivación que la genera. Más aún, en caso que la generación sea ambigua, queremos obtener dos derivaciones distintas para reflejar esa ambigüedad. Para ello hemos implementado un primer paso de normalización de la gramática, luego aplicamos el algoritmo CYK adaptado, y luego manipulamos las derivaciones obtenidas para que correspondan a la gramática previa al paso de normalización. En los siguientes puntos esbozamos cada uno de esos pasos.

[13]http://www.seleniumhq.org/

5.5.3. Normalización

Para hacer más eficientes los pasos posteriores, comenzamos el proceso limpiando la gramática: primero se eliminan las variables no fructíferas (las que no generan ninguna palabra), y luego las no alcanzables (las que nunca intervienen en la generación de una palabra del lenguaje). A continuación, simplificamos las reglas garantizando que sus partes derecha tengan a lo sumo 2 símbolos. Esto se puede lograr fácilmente introduciendo $n-2$ variables auxiliares por cada regla cuya parte derecha tenga $n > 2$ símbolos, y modificando dicha regla de forma conveniente.

Pre-calculamos qué variables pueden generar la palabra vacía, y nos guardamos con qué derivaciones lo pueden hacer (a lo sumo nos guardamos 2 derivaciones, pues no necesitamos más y podría haber infinitas). Típicamente, en los procesos de normalización de las gramáticas se eliminan las reglas de la forma $X \to \varepsilon$ y el caso de la palabra vacía se trata a parte. Sin embargo, nosotros mantenemos estas reglas.

A continuación modificamos la gramática para impedir que haya derivaciones cíclicas de la forma $X \Rightarrow^* X' \Rightarrow^* X'' \Rightarrow^* X''' \Rightarrow^* \ldots \Rightarrow^* X$. Este tipo de derivaciones son indeseables para aplicar CYK, ya que no permiten la formulación recursiva del algoritmo. Para solventar este problema, colapsamos los ciclos: sustituimos todas las variables $X, X', X'', X''', \ldots$ a una nueva variable auxiliar $A_{\{X,X',X'',X''',\ldots\}}$ que no pueda ciclar. Para obtener cada uno de los grupos que queremos colapsar, consideramos el grafo definido de la siguiente manera: los nodos son variables de la gramática, y hay una arista dirigida del nodo X al Y si y solo si hay una regla de la forma $X \to \alpha Y \beta$ y además $\alpha \Rightarrow^* \varepsilon$ y $\beta \Rightarrow^* \varepsilon$. Luego, sobre este grafo calculamos las componentes fuertemente conexas [tro,], ya que cada una de estas componentes corresponde a un subconjunto maximal de variables que pueden realizar derivaciones cíclicas. Cada componente fuertemente conexa se colapsa a una nueva variable auxiliar.

5.5.4. CYK

En nuestro proceso de normalización no hemos eliminado las reglas de la forma $X \to Y$, y por ello debemos proceder con cuidado cuando adaptamos la formulación recursiva de CYK a un cálculo iterativo de programación dinámica. Por ejemplo, dada la presencia de la regla $X \to Y$, primero es necesario realizar los cálculos sobre Y y, una vez concluidos, se puede proceder con los de X. Para solventar esta

dificultad, CYK se debe calcular iterativamente siguiendo una ordenación topológica de las variables [PEARCE and KELLY, 2006].

Por último, el algoritmo CYK tradicional calcula únicamente un booleano, es decir, si existe o no una derivación. Nosotros cambiamos eso para que se almacenen las derivaciones concretas (a lo sumo guardamos 2), en lugar de solo su existencia o no.

5.5.5. Derivaciones

Una vez concluidos los pasos previos, debemos modificar las derivaciones obtenidas para que correspondan a las que se podían realizar con la gramática original, previa a la normalización. Para ello primero debemos localizar las ocurrencias de variables auxiliares introducidas cuando colapsamos los ciclos, y sustituirlas por la subderivación correspondiente. En segundo lugar, debemos quitar las variables auxiliares introducidas cuando limitamos las partes derecha de regla a 2 símbolos como máximo.

Capítulo 6

Gestión

6.1. Metodología y rigor

6.1.1. Metodología Ágil

Hemos utilizado la metodología *scrum* de *agile* en la cual se empieza el proyecto o producto con el mínimo requerimiento para su funcionamiento y de forma iterativa se le añade valor y funcionalidad. Cada iteración es un intervalo de tiempo conocido como *sprint*.

Además de *scrum* hemos utilizado la metodología *kanban*, que nos ha aportado una pizarra donde las tareas están ordenadas y priorizadas en distintas columnas, las cuales son *backlog*, *ready*, *doing* y *done*, que simbolizan el listado de tareas que quedan por hacer de todo el proyecto, listado de tareas que se hacen en esta sprint, listado de tareas que se están haciendo y listado de tareas hechas. En principio las tareas en *backlog* están ordenadas por prioridad/necesidad para el producto. Al principio de cada *sprint* se seleccionan unas cuantas y se valoran por puntos de dificultad, lo cual nos ha ayudado a saber cuanto trabajo se estaba procesando por *sprint*. Esta metodología se suele usar para organizar grupos de programadores de entre 7 y 9 programadores, pero también es bueno para grupos pequeños como el nuestro ya que nos ha servido para llevar un control del avance del proyecto.

Nuestras *sprints* han sido de una semana. Al finalizar la semana se ha

informado al director del estado del proyecto. Junto con él, se han seleccionado las tareas de la semana siguiente. Cada dos semanas se ha hecho una demostración presencial con el director para constatar el trabajo realizado durante las dos semanas anteriores.

6.1.2. Herramientas de seguimiento

El contacto entre director y programador se ha llevado a cabo vía mail, pero en todo momento el director ha podido ejecutar la máquina de pruebas en su dispositivo. Para el control del proyecto a nivel de programación o control de versiones se ha usado un servicio *git*.

6.1.3. Validación

La validación que ha tenido lugar cada dos *sprints* se ha hecho en el servidor de desarrollo, el cual posee una configuración lo más parecida posible a la configuración que tiene el servidor de producción, donde va la aplicación al finalizar el proyecto y en el cual se hacen las ultimas comprobaciones.

Además de las comprobaciones manuales hemos usado la práctica de integración continua, que consta de proveer al modelo de unos tests dirigidos que comprueban que todos los procesos son ejecutables y que retornan lo necesario.

6.2. Planificación temporal

Hemos dividido el proceso de reconstrucción en diversas fases o hitos, que a su vez han sido divididos en pequeñas tareas. Se pretendía que cada una de estas fases fuesen funcionales y el valor de la aplicación aumentase de forma considerable fase a fase. De modo que siempre que se terminase una fase el producto fuera completamente testeable y por lo tanto funcional en todos los sentidos.

6.2.1. Planificación temporal inicial

- Antes de empezar: 2 semanas
  - Estudiar el viejo código e informarse de las necesidades de cambio.
  - Estudiar las posibles soluciones aplicables: Tecnologías y lenguajes.
- Empezar el proyecto: 3 semanas
  - Crear un entorno de desarrollo.
  - Crear una máquina de pruebas donde testear.
  - Inicializar el proyecto web vacío.
- Diseñar front-end y back-end: 3 semanas
  - Guias de estilos.
  - Diseño back-end.
  - Diseño front-end.
- GEP: 6 Semanas
  - Alcance y contextualización.
  - Planificación temporal.
  - Gestión económica y sostenibilidad.
  - Presentación preliminar.
  - Presentación oral y documento final.
- Creamos una primera versión del nuevo Racso: 3 semanas
  - Añadimos la primera entidad llamada Ejercicios.
  - Entidad editable desde el back-end.
  - Entidad tratable desde el front-end.
- Añadimos el resto de entidades: 4 semanas
  - Tipos de ejercicios.
  - Lista de ejercicios.
  - Alumnos.

  - Exámenes.

- Desplegar el proyecto en la máquina de producción. 1 semana
- Documentar y generar un reporte para que los profesores sepan como administrar el panel. 1 semana

Cada hito de esta lista se ha provisto de un juego de pruebas o test con el que verificar que se puede pasar al siguiente hito (a excepción de la documentación).

6.2.2. Descripción de las tareas

Antes de empezar

En esta fase del proyecto lo que se ha pretendido es entender las necesidades del Racso, cuál era el estado de la aplicación y qué requisitos tenía que cambiar o mejorar. Se han estudiado las posibles soluciones que mejor se adaptan a las necesidades. Todo esto fue un estudio previo que ayudó a aclarar cuál era el contexto y el alcance del proyecto. Los tiempos empleados para cada subtarea los podemos ver en la tabla 6.1.

Tarea	Duración
Estudiar código de la vieja versión e informarse de las necesidades de cambio.	16h
Estudiar las posibles soluciones aplicables: Tecnologías y lenguajes.	16h
Total	32h

Cuadro 6.1: Tareas hito “ Antes de empezar ”

Empezar el proyecto

Debido al ajustado tiempo de realización del proyecto se decidió empezar a definir el entorno de trabajo y pruebas antes de tener claro cual sería el alcance y la

contextualización. Los costes de realizar esta tarea previamente son bajos y es algo que podíamos tener preparado para cuando tuviéramos claro el resto del proyecto.

Constaba de 3 sencillas fases en las que simplemente iniciamos el entorno de desarrollo, la máquina de pruebas e inicializamos el proyecto vacío. Podemos ver la duración de cada subtarea en la tabla 6.2.

Tarea	Duración
Crear un entorno de desarrollo.	10h
Crear una máquina de pruebas donde testear.	15h
Inicializar el proyecto web vacío.	8h
Total	32h

Cuadro 6.2: Tareas hito “ Empezar el proyecto ”

Diseñar front-end y back-end

Mientras se estudiaba el contexto y el alcance del proyecto se definió y diseñó una interfaz que hiciera más útil la aplicación, esto nos ayudó a definir el alcance.

Podemos ver la duración de las subtareas de diseño front-end y back-end en la tabla 6.3.

GEP

Durante esta fase pretendíamos definir el contexto de la aplicación y el alcance de la misma, estudiar los costes, las metodologías que hemos usado, la forma de trabajar, el proceso de creación, el impacto, etc. Todo ello era necesario antes de ponerse a trabajar, para poder realizar una tarea precisa y sin sorpresas.

Este trabajo se usó para mostrarle a los profesores qué tareas se iban a realizar durante el proceso.

Tarea	Duración
Guías de estilos	20h
Diseño back-end	15h
Diseño front-end.	15h
Total	50h

Cuadro 6.3: Tareas hito " Diseñar front-end y back-end "

Las horas realizadas para cada subtareas del GEP las podemos ver en la tabla 6.4.

Tarea	Duración
Alcance y contextualización	18h
Planificación temporal	18h
Gestión económica y sostenibilidad	18h
Presentación preliminar	18h
Presentación oral y documento final	18h
Total	90h

Cuadro 6.4: Tareas hito " GEP "

Creamos una primera versión del nuevo Racso

Después de tener claro qué tecnologías íbamos a usar, de tener un diseño en mente y de tener un entorno donde poder trabajar empezamos con una pequeña versión del proyecto la cual sería una base para todo lo que vendría después. La duración de las subtareas de la creación de la primera versión está indicada en la tabla 6.5.

Tarea	Duración
Añadimos la primera entidad llamada Ejercicio	40h
Entidad editable desde el back-end	32h
Entidad tratable desde el front-end	30h
Total	102h

Cuadro 6.5: Tareas hito " Creamos una primera versión del nuevo Racso "

Añadimos el resto de entidades

Después de crear y testear una primera versión dónde ya tenemos el núcleo de la aplicación con una primera clase del modelo introducimos el resto del modelo usando este núcleo. El tiempo requerido para añadir cada una de las Entidades lo indicamos en la tabla 6.6.

Tarea	Duración
Tipos de ejercicios	32h
Lista de ejercicios	32h
Alumnos	32h
Exámenes	32h
Total	128h

Cuadro 6.6: Tareas hito " Añadimos el resto de entidades "

Desplegar el proyecto en la máquina de producción

En esta fase pretendíamos tener el proyecto completamente funcional, pero en el servidor de desarrollor, así que lo que teníamos que hacer era moverlo a la máquina de producción y hacer pruebas con ello. En la tabla 6.7 podemos ver el

tiempo requerido que se había estimado para desplegar el proyecto en la máquina de producción.

Esta es una de las fases que cambiamos en el proyecto final como veréis especificado en el apartado 6.2.3

Tarea	Duración
Desplegar el proyecto en la máquina de producción	8h
Tests en producción	26h
Total	34h

Cuadro 6.7: Tareas hito " Desplegar el proyecto en la máquina de producción "

Documentar y generar un reporte para que los profesores sepan como administrar el panel

Por último y si las pruebas salían satisfactorias se generaría un documento dónde se explicaría a los futuros programadores y usuarios de la aplicación como usar y mantener el Racso. La duración estimada de la tarea de documentación la podemos ver en la tabla 6.8.

Esta es una de las fases que cambiamos en el proyecto final como veréis especificado en el apartado 6.2.3

Tarea	Duración
Documentar y generar un reporte para que los profesores sepan como administrar el panel	32h
Total	32h

Cuadro 6.8: Tareas hito " Documentar y generar un reporte para que los profesores sepan como administrar el panel "

Total horas tareas

El total de horas realizadas y estimadas en tareas y subtareas lo podemos ver en la tabla 6.9.

Tarea	Duración
Antes de empezar	32h
Empezar el proyecto	32h
Diseñar front-end y back-end	50h
GEP	102h
Creamos una primera versión del nuevo Racso	90h
Añadimos el resto de entidades	128h
Desplegar el proyecto en la máquina de producción (Estimada)	34h
Documentar y generar un reporte para que los profesores sepan como administrar el panel (Estimada)	32h
Total	500h

Cuadro 6.9: Tareas hito " Total horas tareas "

6.2.3. Planificación temporal final - Cambios respecto a la planificación inicial

Durante el proceso de desarrollo nos dimos cuenta de que en la fase de Estudio previo no se había estimado bien el tiempo que se tardaría en copiar algunas de las funcionalidades, debido a que algunas de estas funcionalidades estaban ofuscadas en el código y era muy difícil su detección. Se tomó la decisión de aumentar las horas de programación durante la fase de Añadir el resto de entidades, concretamente para la entidad examen. Se aumentó el plazo en dos semanas.

Por otro lado las prioridades del proyecto cambiaron, no había tanta prisa por salir a producción para este cuatrimestre y por lo tanto se decidió aprovechar el

tiempo en desarrollar más cosas de cara a la administración, dotarla de más detalles y ayudas, aumentando el proceso de desarrollo otras dos semanas más.

La duración aproximada inicial del proyecto era de 19 semanas, desde el 15 de enero de 2016 hasta el 27 de mayo de 2016, día escogido por la universidad para presentar el turno de lectura.

La duración final del proyecto ha sido de 22 semanas, desde el 15 de enero de 2016 hasta el 27 de mayo de 2016 como podemos ver en el diagrama de Gantt de la siguiente sección.

6.2.4. Diagrama de Gantt

El diagrama Gantt ha sido separado en dos partes para una mejor visualización. Una parte inicial de 12 semanas 6.1 y una segunda parte de 10 semanas 6.2.

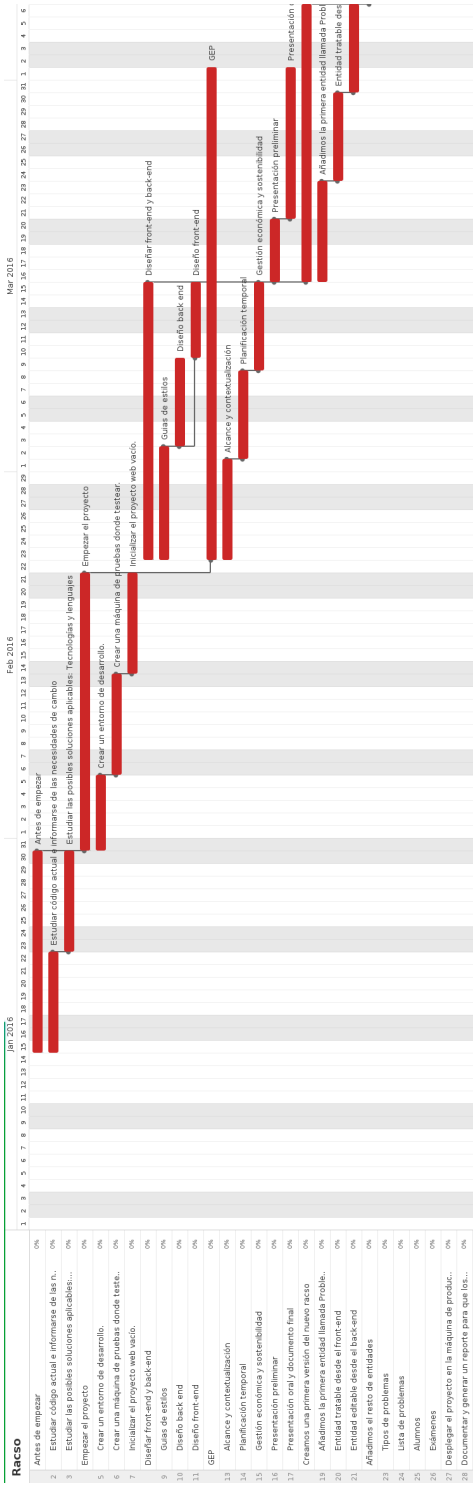

Figura: 6.1: Diagrama de Gantt 1

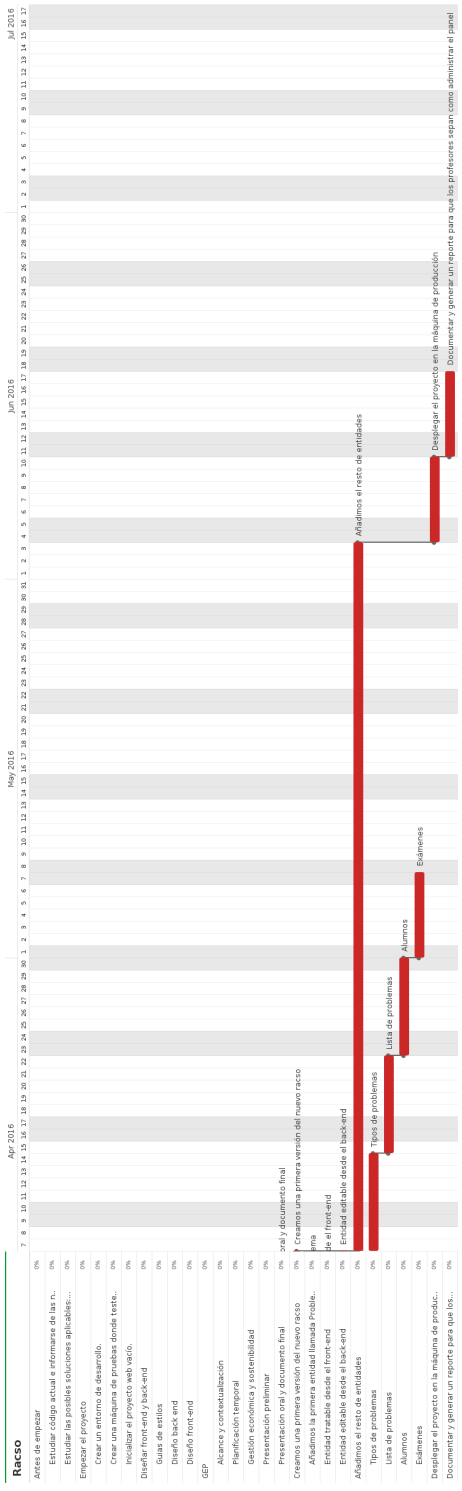

Figura: 6.2: Diagrama de Gantt 2

6.2.5. Distribución del esfuerzo

La distribución del esfuerzo de las horas realizadas por cada rol lo podemos encontrar en la tabla 6.10.

Rol	Horas	Precio hora	Precio total
Director del proyecto	50h	50€/h	2500€
Programador web senior	350h	35€/h	12250€
Diseñador	50h	40€/h	2000€
Tester	50h	20€/h	1000€
Total	500h		17800€

Cuadro 6.10: Tareas hito " Distribución del esfuerzo "

6.3. Gestión económica y recursos

6.3.1. Identificación de los costes

Los costes de la aplicación vienen principalmente definidos por costes de recursos humanos, costes de hardware y costes de software. Entre los costes de hardware identificaremos costes de compra y costes de mantenimiento.

En el proyecto se han intentado usar el máximo de herramientas de software libre o cuentas en servicios gratuitos para abaratar costes como explicaremos en el apartado Costes de software.

Costes de recursos humanos

Estos costes los detallaremos en función de los actores implicados y el tiempo requerido por las tareas en las que están asignados en el Gantt (Sección 6.2.4

Diagrama de Gantt) Los costes de recursos humanos los podemos ver en la tabla 6.11.

Rol	Horas	Precio hora	Precio total
Director del proyecto	50h	50€/h	2500€
Programador web senior	350h	35€/h	12250€
Diseñador	50h	40€/h	2000€
Tester	50h	20€/h	1000€
Total	500h		17800€

Cuadro 6.11: Costes " Costes de recursos humanos "

Costes de recursos de hardware

En la tabla 6.12 mostramos el coste de las máquinas, así como el de su mantenimiento durante por lo menos 5 años.

Producto	Precio	Vida útil	Amortización
Servidor	350€	5 años	70€
Electricidad	3066€	5 años	613,2€
Mantenimiento del servidor	200€	5 años	40€
MacBook Air	1349€	5 años	269,8€
Total	4965€		993€

Cuadro 6.12: Costes " Costes de recursos de hardware "

Costes de recursos de software

Gran parte del software usado, como Laravel, Atom[1] y Gimp[2], son de código abierto y por lo tanto a coste 0. Por otro lado tenemos los servicios online de empresas como Asana, en la cual se incluyen los servicios de Asana[3] y Instagantt[4] y de empresas como Allao Systems, la cual nos ofrece los servicios de Gogs[5] y Waken[6], todos ellos gratuitos. Por último, el único servicio de pago que usamos es el servicio de laracast, el cual nos ofrece toda una documentación y buenas prácticas sobre Laravel. Podemos ver los detalles en la tabla 6.13.

Producto	Precio	Vida útil	Amortización
Gogs	0€	3 años	0€
Wekan	0€	3 años	0€
Asana	0€	3 años	0€
Instagantt	0€	3 años	0€
Sharelatex	0€	3 años	0€
Laravel	0€	3 años	0€
Laracast	86€	1 años	86€
OSX Yosemite	31€	3 años	0€[7]
Atom	0€	1 años	0€
GIMP	0€	1 años	0€
Total	86€		86€

Cuadro 6.13: Costes " Costes de recursos de software "

[1]IDE de desarrolllo.
[2]Editor de imágenes.
[3]Control de un proyecto y sus tareas.
[4]Generar un gantt a partir de las tareas de asana.
[5]Controlador de versiones git.
[6]Pizarra kanban.
[7]Incluido en el MacBook Air.

Costes de imprevistos durante el desarrollo

Debido a la naturaleza del proyecto en la cual estamos reutilizando parte de un modelo anterior, su infraestructura y de la cual no hay una documentación clara, es habitual que surjan pequeños retrasos e imprevistos. Con el fin de que esto no supusiera un verdadero problema se decidió incrementar en un 10 % el presupuesto.

Presupueto total

En la tabla 6.14 podemos ver el presupuesto total de la reconstrucción.

Concepto	Costes amortizados
Coste de recursos humanos	17800€
Coste de hardware	993€
Coste de software	86€
Total (sin imprevistos)	18879€
Coste de imprevistos	10 %
Total	20766,9€

Cuadro 6.14: " Presupueto total "

Control de gestión

En general el control del proyecto se ha llevado a cabo mediante el cumplimiento de los tiempos en el diagrama de Gantt especificado en la planificación temporal. Además gracias a la metodología de trabajo *scrum* al finalizar cada sprint se ha revisado qué tareas se han logrado durante la semana y cuáles no, con el fin de detectar retrasos en el proyecto. Cuando se han detectado más de 2 tareas no logradas por *sprint* se ha hablado con el director para verificar los tiempos. De todos modos se dictó una serie de medidas y consideraciones para poder adaptar estos im-

previstos de la mejor forma posible al proyecto, evitando así su ralentización y por tanto el aumento de los costes:

1. La fecha de finalización del proyecto no se puede posponer nunca.
2. Se dará prioridad a las funcionalidades existentes en el proyecto actual con el fin de detectar los posibles problemas a tiempo y adaptarse lo antes posible al calendario.
3. Si alguna funcionaliad o imprevisto se detecta tarde se valorará su necesidad y se cambiará por otra de igual valor o menor en el backlog de la pizarra kanban
4. Con funcionalidades o imprevistos con un valor importante en la aplicación y que aparezcan cuando el proyecto esté muy avanzado, se tratará de reutilizar el código del antiguo RACSO encapsulado de forma que en el futuro se pueda realizar una tarea de limpieza.

Por último como ya hemos podido observar en el apartado 6.3.1 tomamos una medida de previsión para poder adaptarnos en cuanto al presupuesto en caso de necesitar contratar algún tipo de software extra, contratar más horas de programación o diseño.

6.4. Sostenibilidad y compromiso social

6.4.1. Sostenibilidad económica

El proyecto se va a desarrollar en el ámbito académico con el fin de mejorar la infraestructura de la universidad. Gracias a que algunos de los alumnos desean mejorar este ámbito utilizan sus créditos en forma de trabajo. Este es el caso de un alumno que desea realizar su TFG mejorando el Racso. Gracias a esto la universidad solo tendrá que pagar los costes de mantenimiento y esto hace que los costes sean increíblemente bajos y el precio sea muy competitivo.

Pese a que la universidad no tiene que competir con empresas sí compite con otras universidades por reconocimiento. Tener herramientas nuevas y actualizadas

como lo son: el Moodle, el Racso y el Jutge, sobre todo las dos últimas, que están abiertas a personas externas a la universidad, hace que gane un reconocimiento muy valioso el cual le puede permitir una mejor posición a la hora de recibir ayudas del estado, más alumnos y una mejor infraestructura.

6.4.2. Sostenibilidad social

El Racso es una máquina muy usada por las asignaturas de Teoría de la Computación y Compiladores. Con el tiempo la aplicación queda des-actualizada y requiere de mejoras, el profesorado no dispone de tiempo ni de herramientas suficientes para mejorar el Racso. Por ello este proyecto es una gran ayuda que empieza a ser necesaria para que las asignaturas puedan introducir nuevos ejercicios, métodos de evaluación y métodos de control sobre los alumnos, que hasta ahora no tenían y que empezaban a ser necesarios.

6.4.3. Sostenibilidad ambiental

El consumo de los recursos durante el desarrollo del proyecto han sido bajos ya que no ha sido menester ningún hardware especial ni para desarrollar ni para mantener el producto. Se reutilizará la máquina actual del Racso, gestionada por RDlab [8] y la red de la universidad para conectar la máquina con los usuarios.

6.4.4. Tabla de sostenibilidad

Tabla de sostenibilidad 6.15.

Sostenibilitat	Económica	Social	Ambiental
Planificación	Sección 6.4.1	Sección 6.4.2	Sección 6.4.3
Valoración	8	7	5

Cuadro 6.15: “ Tabla de sostenibilidad ”

[8] https://rdlab.cs.upc.edu/

6.5. Identificación de leyes y regulaciones

Hay dos leyes y regulaciones a tener en cuenta en este proyecto: la ley de protección de datos y la nueva ley europea de cookies.

Para cumplir la ley de protección de datos, los usuarios (como el nombre, la contraseña o el correo) nunca serán públicos y las contraseñas se guardarán de forma cifrada.

La ley de cookies requiere que se avise al usuario en el caso de utilizar cierto tipo de cookies, aquellas que guardan cierta información del usuario y esta no es imprescindible para el funcionamiento de la aplicación. En nuestro caso solo usamos cookies para identificar a nuestros usuarios. No es un caso para el que requiramos advertir a los usuarios.

Capítulo 7

Resultados

7.1. Competencias técnicas

- CCO1.1: Evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan llevar a su resolución, y recomendar, desarrollar e implementar la que garantice el mayor rendimiento de acuerdo a los requisitos [Un poco].

  Se han evaluado múltiples problemas durante el desarrollo. La mejora de tiempo en el proceso de evaluación de un ejercicio disminuyendo el tiempo de respuesta del servidor. El proceso de comunicación cliente-servidor y el proceso de comunicación servidor-servidor ha sido un problema para el cual hemos tenido que definir un pequeño protocolo de comunicación en el que los implicados se identifican previamente.

- CCO1.2: Demostrar los conocimientos de los fundamentos teóricos de los lenguajes de programación y las técnicas de procesamiento léxico, sintáctico y semántico asociadas, y saber aplicarlas para la creación, el diseño y el procesamiento de lenguajes. [Un poco].

  Hemos corregido pequeños errores en los parsers que resaltan las palabras clave en el editor de texto que se utiliza para hacer las entregas de los ejercicios. Además hemos añadido la mini-aplicaciones al contexto del cliente que ayuda los usuarios con los ejercicios de gramáticas y sus resultados.

- CCO1.3: Definir, evaluar y seleccionar plataformas de desarrollo y producción

de hardware y software para el desarrollo de aplicaciones y servicios informáticos de diversa complejidad. [En profundidad].

Como se ha explicado en los capítulos 4 y 5 se han seleccionado plataformas de desarrollo y plataformas de producción así como los scripts para aprovisionar estas máquinas para sus respectivas tareas. Se ha diseñador el sistema como un sistema de múltiples servicios: servicios para el procesamiento de datos y servicios para mejorar la interfaz. Se ha dotado a la plataforma de unos tests para comprobar su estructura y correcto funcionamiento en cualquier momento. En general un ecosistema muy complejo que es capaz de autoevaluarse y lanzarse a producción en pocos pasos.

- CCO2.3: Desarrollar y evaluar sistemas interactivos y de presentación de información compleja, y su aplicación en la resolución de problemas de diseño e interacción persona-computador. [Bastante]

  La cantidad de datos que se muestra en muchas de las pantallas del Racso es muy amplia, compleja y valiosa. En la mayoría de casos simplemente se ha recolocado de forma que cupiera el máximo de información posible en el lugar donde es más imprescindible. Se han tenido en cuenta diversos tamaños de pantalla para que no se perdiera información en pantallas más pequeñas y siempre se pueda disponer de toda la información fácilmente. Se han dado herramientas para listar o paginar la información, así como para buscar entre las grandes cantidades de datos. Se le han puesto facilidades a los administradores como botones de acción rápida para copiar, editar o eliminar elementos.

- CCO3.1: Implementar código crítico siguiendo criterios de tiempo de ejecución, eficiencia y seguridad [Bastante]

  La corrección de un ejercicio es un proceso en el que intervienen pequeñas partes críticas de identificación para el correcto funcionamiento de la aplicación, sobre todo para evitar pirateos de la información. Cada entrega de cada ejercicio es única para cada usuario.

  La seguridad en el momento de un examen se extrema todavía más con comprobaciones constantes a nivel de IP entre los usuarios.

7.2. Conclusiones

Estamos muy contentos con el desarrollo del Racso. Se trata de una aplicación vital para el funcionamiento de ciertas asignaturas y conseguir desarrollar algo así de importante es muy valioso para la carrera de cualquier programador. Es un proyecto con cierta dimensión que a simple vista no se ve, pero que por dentro está repleto de pequeñas cosas que hay que tener en cuenta.

Hemos terminado con el desarrollo que se planteaba en un principio, pero el Racso es una herramienta que no parará de crecer nunca. Siempre se pueden añadir nuevas funcionalidades o mejorar las que ya posee, a nivel de usabilidad, eficiencia o seguridad. Las posibles tareas a hacer en un proyecto como este son casi infinitas. Y mientras se desarrolla, el propio programador se da cuenta de eso y a veces le es complicado saber dónde parar y dónde continuar.

7.3. Futuras implementaciones

A continuación listaremos una serie de funcionalidades y características que podrían ser implementadas como futuras mejoras para el Racso:

- Los alumnos pueden comentar y debatir sobre ejercicios entregados por otros alumnos.
- Aumentar la jerarquía de tipos de usuarios. Sobre todo para distintos tipos de administradores.
- Añadir estadísticas al panel de administración.

  Estadísticas de últimas entregas y nota media.

  Estadísticas de media de entregas por lista. Para cada lista de ejercicios la media de entregas de cada ejercicio.
- Añadir métodos para evitar sobrecargas de corrección de ejercicios. Levantado automático de servicios en caso de ser necesario.
- Unificar código JavaScript en un framework como Vue.js [1].

[1] Framework JavaScript.

- Permitir a cualquier usuario añadir nuevos ejercicios.

Bibliografía

[tro,] Tarjan's algorithm. http://www.programming-algorithms.net/article/44220/Tarjan's-algorithm. [Online; accedido el 21-Jun-2016].

[int, 2014] (2008-2014). International Olympiad in Informatics. http://www.ioinformatics.org/index.shtml. [Online; accedido el 20-Jun-2016].

[cod, 2016a] (2010-2016a). Codeforces. http://codeforces.com/. [Online; accedido el 20-Jun-2016].

[pat, 2014] (2014). Understanding Design Patterns in Laravel. https://www.dunebook.com/understanding-design-patterns-in-laravel/. [Online; accedido el 16-Jun-2016].

[mvc, 2014] (2014). Understanding Model-View-Controller. http://book.cakephp.org/2.0/en/cakephp-overview/understanding-model-view-controller.html. [Online; accedido el 21-Jun-2016].

[cod, 2016b] (2016b). Codewars. http://www.codewars.com/. [Online; accedido el 20-Jun-2016].

[hac, 2016] (2016). HackThisSite. https://www.hackthissite.org. [Online; accedido el 20-Jun-2016].

[tue, 2016] (2016). Tuenti Challenge 6. https://contest.tuenti.net/. [Online; accedido el 20-Jun-2016].

[uva, 2016] (2016). UVa online judge. https://uva.onlinejudge.org/. [Online; accedido el 20-Jun-2016].

[wap, 2016] (2016). Wappanalyzer. https://wappalyzer.com/. [Online; accedido el 20-Jun-2016].

[Documentation, 2014] Documentation, L. (2014). IoC Container. `https://laravel.com/docs/4.2/ioc`. [Online; accedido el 16-Jun-2016].

[GOEDEL, 2006] GOEDEL, K. (2006). *SOBRE PROPOSICIONES FORMALMENTE INDECIBLES DE LOS PRINCIPIA MATHE MATICA Y SISTEMAS AFINES*. KRK EDICIONES, C Álvarez Lorenzana Nº 27, 33006, Oviedo.

[Har-Peled and Parthasarathy, 2009] Har-Peled, S. and Parthasarathy, M. (2009). CYK Parsing Algorithm. `https://courses.engr.illinois.edu/cs373/sp2009/lectures/lect_15.pdf`. [Online; accedido el 21-Jun-2016].

[Nelson, 2016] Nelson, R. C. (2016). Context-Free Grammars. `https://www.cs.rochester.edu/~nelson/courses/csc_173/grammars/cfg.html`. [Online; accedido el 21-Jun-2016].

[PEARCE and KELLY, 2006] PEARCE, D. J. and KELLY, P. H. J. (2006). A Dynamic Topological Sort Algorithm for Directed Acyclic Graphs. `https://www.doc.ic.ac.uk/~phjk/Publications/DynamicTopoSortAlg-JEA-07.pdf`. [Online; accedido el 21-Jun-2016].

[Symfony, 2016] Symfony (2016). Why should I use a framework? `http://symfony.com/why-use-a-framework`. [Online; accedido el 16-Jun-2016].

Agradecimientos

Quisiera agradecer la idea para este proyecto a la asignatura obligatoria de la especialidad de computación Teoría de la Computación ya que gracias a esta asignatura tuve contacto con el Racso por primera vez. Quisiera agradecérselo también a la asignatura de Lenguajes de programación porque me incitó a investigar otros lenguajes de programación, a aprender a mirar más allá, compararlos y poder centrarme en su utilización. Quisiera agradecérselo a mi Director de proyecto y a mi Ponente ya que sin ellos este trabajo no hubiera sido posible. Un agradecimiento sincero a todos aquellos compañeros míos de la universidad que se han preocupado por el avance del proyecto. Y por último, y también más personal, quiero agradecérselo a mi familia y a mi novia, que me han apoyado en todo momento y me han acompañado y ayudado a seguir incluso en los momentos en los que ni yo mismo sabía muy bien cómo.

A todos vosotros, gracias.